UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/714,198	11/14/2003	Long Li	42P17832	2667

8791 7590 09/03/2008

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
09/03/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/714,198	LI ET AL.	
	Examiner	Art Unit	
	TUAN A. VU	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *14 July 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-9,11-19,21,24-26,29,30 and 34-36* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-9,11-19,21,24-26,29,30 and 34-36* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 7/14/08.

As indicated in Applicant's response, claims 1, 11, 13, 21, 24-26, 29-30, 34 have been amended, and claims 10, 20, 22-23, 27-28, 31-33 are canceled. Claims 1-9, 11-19, 21, 24-26, 29-30, 34-36 are pending in the office action.

Claim Rejections - 35 USC § 112

2. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

3. Claims 21, 24-25, 26, 29-30 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention.

Specifically, claim 21, 26 recite 'a program-thread-partition loop' in light of 'critical sections of the program-thread-partition loops'; this *program-thread-partition loop* entity is not disclosed in the Specifications in terms as to enable one of ordinary skill in the art to perceive how a loop is created to enable program threads to be partitioned. The phrase 'program-thread-partition' entails a partitioning connotation, and a loop to effectuate such connotation would have to be described in order for one to have grasp of this 'program thread partition loop' effect; regarding which phrase, the Specifications is completely silent. Further, the *critical sections* of the 'program-thread-partition loops' is not provided with clear support from the Disclosure. That

is, regarding the critical sections in Fig. 2B-C (see Specifications pg. 7), data dependency being carried within a thread is effectuating by a 'for' loop; but this alone cannot justify how a thread-partition loop has been implemented, nor can this explain how the *critical sections* are identified within to any such loop, since as depicted in the Figure 2A critical sections belong to the non-partitioned infinite loop 282; i.e. the 'for' loop in Fig. 2B does not contain any *critical sections* per se. The Applicant is not deemed in possession of a loop that supports 'program thread partition' nor in possession of any *identified critical section* residing within such a "program-thread-partition loop"; and claim 21 is rejected for not provided with proper enabling support.

Likewise, claim 26 recites 'program-thread-partition loop', a phrase that is not mentioned once in the Specifications; nor is the 'identified critical sections of the program-thread-partition loops' phrase reasonably and precisely conveyed from the Disclosure; and claim 26 is also rejected for lack of description.

The 'program-thread-partition loop' and 'identified critical sections' would be given weight only to the extent (emphasis added) that *sections* identified from the boundaries checking within a global loop construed from a CFG analysis, require synchronization; whereas said boundaries would convey a form of partitioning of those sections for such synchronizing purpose.

Claims 24-25, 29-30 are rejected for failing to show how a loop to partition program threads is all about, and how 'critical sections' are identified within such loop; absent explicit discussion of the phrase 'program-thread-partition loop' any where in the Disclosure.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1-2, 11-12, 21, 24, 26, 29, 34-35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Sohi et al, "Multiscalar Processors", 1995, ACM 0-89791-698; pp. 414-425 (hereinafter Sohi)

As per claim 1, Sohi discloses a method comprising:

building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop (e.g. *CFG ... entire loop* - sec 2.1, L col. pg. 415);

updating nodes of the CFG loop to enclose identified sections of the sequential application program within corresponding pairs of boundary instructions (e.g. *partitioned ... tasks* – R col. pg 414; *partitioned ... boundaries of a task ... instruction at ... exiting edges* -- R col. pg. 417; *iteration ... sequential instruction stream ... Head and tail ... current tasks ... values are bound to storage ... registers and memory* – pg. 415, bottom R to pg. 416, top L col.); and

modifying the updated nodes of the CFG loop to reduce an amount of instructions between the corresponding pairs of boundary instructions to form a modified CFG loop (e.g. pg. 416, R col. top & middle – Note: *retiring* task with readjusted path based on register value at *head* and *tail* recordings reads on reducing code between boundaries – see sec 3, pg. 419);

generating a plurality of program-thread-partitions from the modified CFG loops (e.g. *task, partitioned, boundaries* – R col. pg. 417 to L col pg. 418);

concurrently executing (sec 3.1.1 pg. 419) a plurality of program-thread-partitions that are generated (e.g. *search ... linked list parallel execution* – bottom R col. pg. 416 to top 417 L col; Fig. 4 - Note: updating and resolving values via traversing a CFG for parallel re-allocating of correctly resolved task each in its runtime own context **reads on** generating of program thread partitions – see *partitioned by the compiler* - R col. pg. 417; *each of which fetches and executes the instructions in its task* – see sec 6, pg. 425; *threads ... multiscalar processor* - pg. 422, R col. top) from the modified CFG loop;

execution of the identified sections of the sequential program among the plurality of concurrently executing program-thread-partitions to ensure that the identified sections are executed in a sequential thread order (e.g. sec 3.1.1 pg. 419).

But Sohi does not explicitly disclose that the identified sections are *critical sections* and, based on which to partition and *synchronizing execution* of those identified critical sections. However, Sohi teaches synchronizing in view of memory accesses to ensure no conflicts and this is strongly suggestive on avoiding contention for variable access (e.g. *offending accesses* – top L col. pg. 420) among instructions being traversed during Sohi's CFG partitioning; hence discloses addressing memory accesses in terms that they are critical as in being potentially problematic to the resource dependency related to the parallel execution of tasks. To that effect it can be reasonable to say that Sohi addresses critical sections; hence, this 'critical section' limitation would have been if not disclosed, then obvious. For one skill in the art at the time the invention was made it would have been obvious to implement the boundary instructions by Sohi so that

regions being identified as register-related accesses (see L col. pg. 416; sec 2.2, pg 417, L col) are critical memory access regions on the CFG in light of the dependency of potential successors with respect to a predecessor in the CFG (e.g. sec 3.1.1, pg. 419-420) based upon which effectuate synchronized access execution thereof, because this would enforce a more proper boundary instructions to address such potential contention between successor and predecessor task or thread in the sections being analyzed in Sohi static setup (see Fig. 2, pg 415; task descriptor - pg. 417-418)

As per claim 2, Sohi (in view of the rationale in claim 1) discloses wherein updating the CFG loop comprises:

selecting an identified critical section of the sequential application program (Fig. 2, pg 415; task descriptor - pg. 417-418); and inserting boundary instructions at a top node and a bottom node of the CFG (e.g. Head, Tail – Fig. 1, pg. 415);

repeating the selecting, inserting and inserting for each identified critical section of the sequential application program (e.g. R col. pg. 417; pg. 415, bottom R to pg. 416, top L col.).

As for the inserting recited as: *inserting an await instruction within a top node of the CFG loop; inserting an advance instruction within a bottom node of the CFG loop*, this is deemed disclosed by Sohi, as follows.

Sohi discloses a multiscalar model of execution uses a CFG to inspect instruction dependency per tasks per processor for re-ordering based on path inspection (see pg. 414, R column; col. 415, R col; re-ordered – sec 4.1) and one such task being inspected is coordinated with bit or mask implemented as an *instruction* at the beginning of a CFG region just so said instruction would indicate a *wait* within a subsequent *forward* step dependent on whether

resources are properly allocated to be released for the awaiting task to continue or *advance* (see *must wait ... to release in order to continue* - pg. 417, R col; pg 418, L col). Hence, Sohi also discloses a wait at the top of a critical section and an advance when resources are correct for resuming a waiting task.

As per claim 11, Sohi discloses an article of manufacture including a machine readable medium having stored thereon instructions which may be used to program a system to perform a method, comprising:

building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop;

updating nodes of the CFG loop to enclose identified sections of the sequential application program within corresponding pairs of boundary instructions; and

modifying the updated nodes of the CFG loop to reduce an amount of instructions between corresponding pairs of bonding instructions to form a modified CFG loop; generating a plurality of program-thread-partitions from the modified CFG loops;

concurrently executing a plurality of program-thread-partitions that are generated from the modified CFG loop;

execution of the identified sections of the sequential program among the plurality of concurrently executing application program-thread-partitions to ensure that the identified critical sections are executed in a sequential thread order.

All of which limitations have been addressed in claim 1.

But Sohi does not explicitly disclose that the identified sections are *critical sections* and, based on which to partition and *synchronizing execution* of those identified critical sections. This limitation has been addressed in claim 1.

As per claim 12, refer to claim 2.

As per claim 21, Sohi discloses a method comprising:

partitioning a sequential application program into a plurality of program-thread-partitions according to a predetermined thread count (e.g. *search ... linked list parallel execution* – bottom R col. pg. 416 to top 417 L col; Fig. 4; see middle R col. pg. 417 – Note: partitioned nodes on CFG via bounding of tasks, leading to reordering for task execution in parallel **reads on** generating of thread partitions according to thread count - see: *threads ... multiscalar processor* - pg. 422, R col. top – Note: 'thread count' not given actual merits – see USC 112, 1st para rejection) of a multi-threaded architecture, each program-thread-partition including a program-thread-partition loop (Note: refer to **USC 112, 1st Rejection**, where *program-thread-partition loop* and *critical sections* are equated to bounded portions of the CFG for which synchronization is needed); and

concurrently executing the plurality of program-thread-partitions (see: *each of which fetches and executes the instructions in its task* – see sec 6, pg. 425; *threads ... multiscalar processor* - pg. 422, R col. top) within a respective thread of the multi-threaded architecture; and

access to identified sections of program-thread-partition loops of the sequential application program among the plurality concurrently executing program-thread-partitions to execute the identified critical sections of the program-thread-partitions loops in sequential thread order (sec 3.1.1 pg. 419).

But Sohi does not explicitly disclose that the identified sections being synchronized are *critical sections and synchronizing access* of said critical sections; however, this limitation has been deemed an obvious limitation as set forth in claim 1.

As per claim 24, Sohi discloses wherein the partitioning of claim 21 comprises:

receiving an indication of the identified critical sections within the sequential application program (*CFG ... entire loop* - sec 2.1, L col. pg. 415);

processing identified critical sections to reduce an amount of code (e.g. *confirm ... correct execution ... correct path ... CFG is resolved incorrect must be squashed ... correct data recovered … retiring the task ... updating the head pointer*– pg. 416, R col. top & middle – Node: eliminate task for which resources are not proper reads on reducing code between boundary of critical sections; sec 3.1 pg. 419) contained within the critical sections of the program-thread-partition loops.

As per claim 26, Sohi discloses an article of manufacture including a machine readable medium having stored thereon instructions which may be used to program a system to perform a method, comprising:

partitioning a sequential application program into a plurality of program-thread-partitions according to identified sections within the sequential application program, each program-thread-partition including a program-thread-partition loop (Note: refer to **USC 112, 1st Rejection**, where *program-thread-partition loop* and *critical sections* are equated to bounded portions of the CFG for which synchronization is needed); and

concurrently executing the plurality of program-thread-partitions within a respective thread of the multi-threaded architecture; and

access to the identified sections among the plurality of concurrently executing program-thread-partitions to ensure that the identified sections of program-thread-partitions are executed in a sequential thread order;

all of which having been addressed in claim 21.

But Sohi does not explicitly disclose that the identified sections are critical sections and synchronizing access of those sections; however, this limitation has been deemed an obvious limitation as set forth in claim 1.

As per claim 29, the use of synchronization regarding contention within critical sections as set forth in claim 1, has rendered the reduction of code amount contained in these critical sections obvious by virtue of the technique of synchronizing, whereby contention due to amount access instructions from threads is reduced.

As per claim 34, Sohi discloses a system comprising: a processor; a memory controller coupled to the processor; and a DDR SRAM memory coupled to the processor, the memory including a compiler is operable to cause

partitioning of a sequential application program into a plurality of program-thread-partitions according to identified sections within the sequential application program to enable

concurrent execution of the plurality of program-thread-partitions within a respective thread of a multi-threaded architecture and

to access to the identified sections among the plurality of concurrently executing program-thread-partitions to ensure that the identified sections are executed in a sequential thread order;

all of which having been addressed in claim 21.

But Sohi does not explicitly disclose that the identified sections are critical sections and synchronizing access of those sections; however, this limitation has been deemed an obvious limitation as set forth in claim 1.

As per claim 35, Sohi discloses wherein the compiler to cause building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop to cause an update of nodes of the CFG loop to enclose identified critical sections of the sequential application program within pairs of boundary instructions (e.g. *partitioned ... tasks* – R col. pg 414; *partitioned ... boundaries of a task ... instruction at ... exiting edges* -- R col. pg. 417; *iteration ... sequential instruction stream ... Head and tail ... current tasks ... values are bound to storage ... registers and memory* – pg. 415, bottom R to pg. 416, top L col) and

to cause modification of nodes of the CFG loop to reduce an amount of instructions (e.g. *confirm ... correct execution ... correct path ... CFG is resolved incorrect must be squashed ... correct data recovered … retiring the task ... updating the head pointer*– pg. 416, R col. top & middle – Node: eliminate task for which resources are not proper reads on reducing code between boundary of critical sections; sec 3.1 pg. 419) between corresponding pairs of bonding instructions to form a modified CFG loop.

6. Claims 3-9, 13-19, 25, 30, 36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Sohi et al, "Multiscalar Processors", 1995, ACM 0-89791-698; pp. 414-425; in view of Vijaykumar T.N., "Compiling for the Multiscalar Architecture", University of Wisconsin, 1998, pp. 1-191 (hereinafter Vijayk).

As per claim 3, Sohi disclose *await* and *advance* instructions as in

reordering identified motion candidate instructions within the nodes of the CFG loop with fixed await instructions and fixed advance instructions (see pg. 414, R column; col. 415, R col; re-ordered – sec 4.1; see *must wait ... to release in order to continue* - pg. 417, R col; pg 418, L col);

but does not disclose wherein modifying (nodes of CFG loop) comprises: *hoisting detected hoist instructions from identified motion candidate instructions within the nodes of the CFG loop using code motions with fixed await boundary instructions into corresponding predecessor basic blocks; and sinking detected sink instructions from identified motion candidate instructions within the nodes of the CFG loop using code motion with fixed advance instructions into corresponding successor basic blocks.*

The *code motion* based on judicious analysis of program resources based on CFG and program resources information collected by a compiler for scheduling a multiscalar ILP execution, as by Sohi, is also disclosed in Vijayk, and according to which, Vijayk discloses successor task or predecessor task susceptible to be rescheduled or stalled based on register data validation or prediction implemented by wait instructions or (resources) forward annotation (Fig. 3-6, pg. 61; Fig. 4-8, pg. 100; sec 4.4.5-4.4.6); and that *code motion* can applied to candidature blocks based on a wait so to be hoisted to a higher layer of CFG critical portion (or basic blocks) without disrupting the structure of the tree basic block dependency (see Fig. 4-9, 4-10, pg. 101-102; Fig. 4-13, pg. 109); and well as sinking of instruction to send them deeper into another successor CFG sub-tree (pg. 102-103; Fig. 4-10; sec 4.5.5 ,pg. 108-110; Fig 4-15, 4-16 pg. 113) to reduce redundant stall cycles. It would have been obvious for one skill in the art at the time the invention was made to implement to wait and resources forward boundary instructions by

Sohi to that Sohi's partitioning of the modified node information based on registers also include code motion as to reduce unused code cycles and improve overall code size by hoisting instructions to where resources can be resolved earlier in the predecessor portion or forwarded into a successor sub-tree as explained in the multiscalar's ILP compiler by Vijayk. One would be motivated to do so because this enable the amount of unused cycles (e.g. in Sohi's predecessor or successor subtrees) be optimized and therey reduce the code between two bounded critical subtree via hoisting and sinking as evidenced by Vijayk in the scheduling of task, resources validation and look-ahead control for loop restructuring (see pg. 60-117)

As per claim 4, Sohi discloses initializing a CFG for code restructuring based on register data validation and tasks reducing or squashing using await for resource resolution; that is, Sohi discloses queue of task order for setting boundary instructions (see queue – R col. bottom pg. 415) and prediction with advance knowledge to reduce stalling of cycles (see Sohi: *minimizing cycles lost* , pg. 421; *predictor ... advance knowledge* - sec 4.2 pg. 422)

but does not specifically disclose wherein hoisting detected hoist instructions with fixed await instructions comprises: 'identifying every instruction within a basic block the CFG loop, excluding await instructions, as a motion candidate instructions; building an inverse graph of the CFG loop; initializing a hoist queue with the basic blocks from the CFG loop, the basic blocks ordered according to a topological order indicated by the inverse graph; hoisting motion candidate instructions of the basic blocks into corresponding predecessor basic blocks until hoist instructions are no longer detected from the identified motion candidate instructions; and hoisting detected hoist instructions from motion candidate instructions within a source basic block of the CFG loop according to a dependence graph of the sequential application program'.

The providing of candidate motion instructions is disclosed in Vijayk using traversal of a forward tree and reverse tree (Fig. 3-13, pg. 79; upward … reverse topological order – pg. 102); and using algorithm to investigate candidate instruction (set as list of task – see Fig. 3-11, pg 75) based on the tree initialization and basic block resource analysis (e.g. register stalls Fig. 4-5, 4-6, pg. 95-96) and dependency of predecessor and successor in order to implement hoisting and sinking (refer to claim 3), i.e. until hoist instructions are no longer detected from the identified motion candidate instructions. It would have been obvious for one skill in the art at the time the invention was made to implement the CFG resources dependency by Sohi so that it is initialized in form of basic blocks with dependency of resources as by Vijayk, as well as task allocation and resolution of dependency based on upward and then reverse traversal, and then predicting the instructions exclusively within the boundary instructions as set forth in claim 1, to promote code motion as set forth in claim 3 above using the teaching by Vijayk; because this would enable Sohi's multiscalar ILP compiler to further achieve code reduction (as set forth in claim 3) and bi-directional establishing of adjusted dependency while searching of candidate instructions to be moved (sinking or hoisting – see Vijayk: pg. 101-103)

As per claim 5, Sohi does not disclose wherein hoisting detected hoist instructions from the motion candidate instructions of the basic blocks comprises:

de-queuing a basic block from the hoist queue as a current block;

computing hoist instructions from the motion candidate instructions based on a dependence graph of the sequential application program;

hoisting the computed hoist instructions into a corresponding basic block; and enqueuing the current block's predecessors from the CFG loop into the hoist queue when a change is detected.

Sohi discloses queue of task order for setting boundary instructions (see queue – R col. bottom pg. 415) and prediction with advance knowledge to reduce stalling (see Sohi: *minimizing cycles lost* , pg. 421; *predictor ... advance knowledge* - sec 4.2 pg. 422) in order to determine how task being initialized in a CFG (see Fig. 2, pg. 415) can be removed to reduce cycle stalling. Based on the code motion technique by Vijayk (refer to claim 3) using the same resource dependency validation and sequencer/predictor as in Sohi and also in view of the dequeuing (i.e. enqueueing at first prior to dequeueing) of task as by Vijayk (see Control Flow heuristics, *dequeue* – Figure 3-10, pg. 74), the de-queuing of a candidate basic block for code motion -- identified as not disclosed in Sohi-- would have been obvious in view of the rationale – using the teaching by Vijayk -- to address traversal of an initialized tree of CFG until hoist instructions are no longer detected from the identified motion candidate instructions (i.e. heuristics for initializing task or candidate instruction as ordered list and dequeuing one by one), in claim 4.

As per claim 6, Sohi does not explicitly disclose identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions; initializing a sink queue with the basic blocks ordered based on a topological order in the CFG loop; sinking detected sink instructions among the basic blocks into corresponding successor basic block until sinking instructions are no longer detected from the identified motion candidate instructions; and reordering detected motion candidates within basic blocks that contain advance instructions according to the dependence graph.

But code motion using a initialized candidate task or basic blocks using a predictor and dependency of register resource allocation based on such candidate instruction or basic block for sinking or hoisting has been addressed in dequeueing/enqueueing limitations as set forth in claim 5 above in light of the rationale of claim 4.

As per claim 7, Sohi does not explicitly disclose wherein sinking detected sink instructions among the basis blocks comprises: de-queue a basic block from the sink queue as a current block; computing sink instructions from motion candidate instructions based on a dependence graph of the sequential application program; sinking computed sink instructions into a corresponding successor basic block; and en-queuing a current block's successors in the CFG loop into the sink queue if a change is detected. But the dequeing and enqueing limitation has been addressed as obvious for both hoisting and sinking as set forth in claim 5 in view of the hoist/sink candidate or listed instructions being hoisted or sunk (*until ... instructions are no longer detected*) of claim 4.

As per claim 8, Sohi discloses reordering the identified motion candidates instructions within basic blocks (e.g. *part of a basic block, a basic block* – sec 2.2, top L col. pg. 415) that contain await instructions based on a dependence graph of the sequential application program (refer to claim 1); but does not disclose initializing a hoist queue with the basic blocks ordered based on a topological order in the CFG loop; identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions and fixed await instructions; hoisting detected hoist instructions among the basic blocks into corresponding predecessor basic blocks until hoist instructions are no longer detected from the identified motion candidate instructions.

But the initializing of basic block being bounded by some fixed AWAIT or ADVANCE instructions for analysis of dataflow so to hoist or sink (until hoist candidate instructions are not longer detected) has been addressed in claim 4.

As per claim 9, Sohi does not explicitly disclose wherein motion candidate instructions hoisted out of an outmost await instruction are no longer treated as motion candidates; and wherein motion candidate instructions out of an outmost advance instruction are no longer treated as motion candidates. But the candidate instructions being hoisted from a initialized candidate list as by Vijayk (until hoist candidate instructions are not longer detected) amount to the subject matter as set forth in claim 4; and would have been obvious using the rationale of claims 4 and 5.

As per claims 13-19, refer to claims 3-9, respectively.

As per claim 25, Sohi does not disclose wherein *code motion* is used to reduce the amount of code contained within the identified critical sections of the thread program loops. But the hoist and sink code motion has been addressed in claim 3 above, using Vijayk.

As per claim 30, refer to claim 25.

As per claim 36, refer to claim 3.

Response to Arguments

7. Applicant's arguments filed 7/14/08 have been fully considered but they are not persuasive. Following are the Examiner's observation in regard thereto.

35 USC § 103 Rejection:

(A) Applicants have submitted that Sohi does not disclose or suggest: 'modifying the updated nodes', plurality of 'program-thread-partitions', or 'generating … program thread partitions'; and

'synchronizing … among the plurality of … program-thread-partitions' as in claim 1 (Appl. Rmrks pg. 13). These are not provided with corresponding rationale and supporting facts showing, each, how the cited portions corresponding thereto would be inapposite or clearly counter-teaching. Nor the arguments point out how exactly each and every prongs in the 103 Rejection is deemed unsupported or improperly established to meet the missing element of the claim. Mere pleading that Applicants have not been able to discern (Appl. Rmrks pg. 14, middle) teachings (from Sohi) does not constitute a clear prima facie in terms as to clearly show how the cited portions (in Sohi) fail to map certain language specific of the claim; and such pleading without facts amounts to mere allegation that a claim is patentable because the Office Action is completely incapable of effectuating any matching from the reference. Thus, Applicants' approach is deemed not commensurate with the specifics set forth in the Office Action and Rejection; hence, according to non-compliant aspect of said pleading with respect the CFR 1.111b, the argument is deemed non-persuasive in overcoming the state of the rejection. Besides, the arguments that apply to a language that is newly added would be deemed largely **moot** because of the effect of the Amended language and a newly implemented form of Rejection.

(B) Applicants submitted that Applicants disagree with the combination of Sohi and VijayKumar as set forth in the rejection of claims 3-9,13-19, 25, 30, 36 (Appl. Rmrks pg. 14 bottom). However, this amounts to mere allegation that overall, the Rejection is not proper; and allegation not accompanied of factual evidences to prove its weight would be treated as no prima facie case for a proper response suitable to overcome a grounds of Rejection.

In all, the claims (including the newly added limitations) will stand rejected as set forth in the above Office Action.

Conclusion

8. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Tuan A Vu/

Primary Examiner, Art Unit 2193

August 31, 2008